Improved Non-Committing Encryption Schemes based on a General Complexity Assumption

Ivan Damgård and Jesper Buus Nielsen

BRICS* Department of Computer Science
University of Aarhus
Ny Munkegade
DK-8000 Arhus C, Denmark
{ivan,buus}@brics.dk

Abstract. Non-committing encryption enables the construction of multiparty computation protocols secure against an *adaptive* adversary in the computational setting where private channels between players are not assumed. While any non-committing encryption scheme must be secure in the ordinary semantic sense, the converse is not necessarily true. We propose a construction of non-committing encryption that can be based on any public-key system which is secure in the ordinary sense and which has an extra property we call *simulatability*. This generalises an earlier scheme proposed by Beaver based on the Diffie-Hellman problem, and we propose another implementation based on RSA. In a more general setting, our construction can be based on any collection of trapdoor permutations with a certain simulatability property. This offers a considerable efficiency improvement over the first non-committing encryption scheme proposed by Canetti et al. Finally, at some loss of efficiency, our scheme can be based on general collections of trapdoor permutations without the simulatability assumption, and without the common-domain assumption of Canetti et al. In showing this last result, we identify and correct a bug in a key generation protocol from Canetti et al.

1 Introduction

The problem of multiparty computation dates back to the papers by Yao [20] and Goldreich et al. [15]. What was proved there was basically that a collection of n players can efficiently compute the value of an n-input function, such that everyone learns the correct result, but no other new information. More precisely, these protocols can be proved secure against a polynomial time bounded adversary who can *corrupt* a set of less than $n/2$ players initially, and then make them behave as he likes. Even so, the adversary should not be able to prevent the correct result from being computed and should learn nothing more than the result and the inputs of corrupted players. Because the set of corrupted players is fixed from the start, such an adversary is called *static* or non-adaptive.

* Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

There are several different proposals on how to define formally the security of such protocols [19, 3, 8], but common to them all is the idea that security means that the adversary's view can be *simulated* efficiently by a machine that has access to only those data that the adversary is *entitled* to know. Proving correctness of a simulation in the case of [15] requires a complexity assumption, such as existence of trapdoor permutations. Later, *unconditionally* secure MPC protocols were proposed by Ben-Or et al. and Chaum et al.[6, 10], in the model where *private* channels are assumed between every pair of players. These protocols are in fact secure, even if the adversary is *adaptive*, i.e. can choose dynamically throughout the protocol who to corrupt, as long as the total number of corruptions is not too large. It is widely accepted that adaptive adversaries model realistic attacks much better than static ones. Thus it is natural to ask whether adaptive security can also be obtained in the computational setting?

If one is willing to trust that honest players can erase sensitive information such that the adversary can find no trace of it, should he break in, then such adaptive security can be obtained quite efficiently [5]. Such secure erasure can be too much to hope for in realistic scenarios, and one would like to be able to do without them. But without erasure, protocols such as the one from [15] is not known to be adaptively secure. The original simulation based security proof for [15] fails completely against an adaptive adversary.

However, in [9], Canetti et al. introduce a new concept called *non-committing encryption* and observe that if one replaces messages on the secure channels used in [6, 10] by non-committing encryptions sent on an open network, one obtains adaptively secure MPC in the computational setting. They also showed how to implement non-committing encryption based on so called common-domain trapdoor permutations. The special property of non-committing encryption (which ordinary public-key encryption lacks) is the following: although a normal ciphertext determines a plaintext uniquely, encrypted communication can nevertheless be simulated with an indistinguishable distribution such that the simulator can later "open" a ciphertext to reveal any plaintext it desires. In an MPC setting, this is what allows to simulate the adversary's view before *and after* a player is corrupted. The scheme from [9] has expansion factor at least k^2, i.e., it needs to send $\Omega(k^2)$ bits for each plaintext bit communicated.

Subsequently, Beaver [4] proposed a much simpler scheme based on the Decisional Diffie-Hellman assumption (DDH) with expansion factor $O(k)$. Recently, Jarecki and Lysyanskaya [17] have proposed an even more efficient scheme also based on DDH with constant expansion factor, which however is only non-committing if the receiver of a message is later corrupted. This is sufficient for their particular application to threshold cryptography, but not for constructing adaptively secure protocols in general.

2 Our Results

In this paper, we first present a definition of *simulatable* public-key systems. This captures some essential properties allowing for construction of non-committing

encryption schemes based on ordinary semantically secure public-key encryption. Roughly speaking, a public-key scheme is simulatable if, in addition to the normal key generation procedure, there is an algorithm to generate a public key, without getting to know the corresponding secret key. Moreover, it must be possible to sample efficiently a random ciphertext without getting to know the corresponding plaintext (we give precise definitions later in the paper)

We then describe a general way to build non-committing encryption from simulatable and semantically secure public-key encryptions schemes. Our method offers a major improvement over [9] in the number of bits we need to send. It may be seen as a generalisation of Beaver's plug-and-play approach from [4]. Beaver pointed out that it should be possible to generalise his approach to more general assumptions than DDH. But no such generalisation appears to have been published before.

The idea that it could be useful to generate a public key without knowing the secret key is not new. It seems to date back to De Santis et al.[12] where it was used in another context. The idea also appears in [9], but was only used there to improve the key generation procedure is some special cases (namely based on discrete logarithms and factoring). Here, we show the following

- From any semantically secure and simulatable public-key system, one can construct a non-committing encryption scheme.
- The scheme requires 3 messages to communicate k encrypted bits, where k is the security parameter. The total amount of communication is $O(k)$ public keys, $O(k)$ encryptions of a k-bit plaintext (in the original scheme), and k bits.
- Only the final k bits of communication depend on the actual message to be sent, and hence nearly all the work needed can be done in a preprocessing phase.

As mentioned, the DDH assumption is sufficient to support this construction. We propose an alternative implementation based on the RSA assumption, which is somewhat slower than the DDH solution[1].

We then look at general families of trapdoor permutations. We call such a family simulatable if one can efficiently generate a permutation in the family without getting to know the trapdoor, and if the domain can be sampled in an invertible manner. Invertible sampling is a technical condition which we discuss in more detail later. All known examples of trapdoor permutations have invertible sampling. Although this condition seems to be necessary for all applications of the type discussed in [9, 4] and here, it does not seem to have been identified explicitly before.

We show that such a simulatable family implies immediately a simulatable public-key system with no further assumptions. The non-committing encryption scheme we obtain from this requires per encrypted bit communicated that we send $O(1)$ descriptions of a permutation in the family and $O(k)$ bits (where the

[1] A proposal with a similar idea for the key generation but with a less efficient encryption operation was made in [9].

hidden constant only has to be larger than 2, and where all bits except one can be sent in a preprocessing phase). With the same assumption, the scheme from [9] requires $\Omega(1)$ permutation descriptions and $\Omega(k^2)$ bits. Moreover, the $\Omega(k^2)$ bits depend on the message communicated and so cannot be pushed into a preprocessing phase. On the other hand it should be noted that the scheme from [9] needs only 2 messages (rather than 3 as in our scheme). It is not known if the same improvement in bandwidth can be obtained with only 2 messages.

Our final main result is an implementation of non-committing encryption based on any family of trapdoor permutations, assuming only invertible sampling, i.e., without assuming full simulatability or the common-domain assumption of [9].

At first sight, this seems to follow quite easily from the results we already mentioned, if we use as subroutine a key generation protocol from [9]. This protocol is based on oblivious transfer and can easily be modified to work based on any family of trapdoor permutations, assuming only invertible sampling. The protocol was intended to establish a situation where a player knows the trapdoor for one out of two public trapdoor permutations, without revealing which one he knows. It turns out that our scheme can start from this situation and work with no extra assumptions.

However, as we explain later, we have identified a bug in the key generation protocol of [9] causing it to be insecure. Basically, there is a certain way to deviate from the protocol which will enable the adversary to find out which of the two involved trapdoors is known to an honest player. We suggest a modification that solves this problem. While the modification is very simple and just consists of having players prove correctness of their actions by standard zero-knowledge protocols, it is perhaps somewhat surprising that it works. Standard rewindable zero-knowledge often cannot be used against an adaptive adversary: the simulator can get stuck when rewinding if the adversary changes its mind about who to corrupt. However, in our case, we show that the simulator will never need to rewind.

We note that the key generation of [9] needs invertible sampling in any case, and thus our assumption of existence of trapdoor permutations with invertible sampling is the weakest known assumption sufficient for non-committing encryption.

3 A Quick and Dirty Explanation

Before going into formal details, we give a completely informal description of some main ideas. Let R, S be the two players who want to communicate in a non-committing way.

First S chooses a bit c and generates a pair of public keys (P_0, P_1) such that he only knows the secret S_c key corresponding to P_c. He sends (P_0, P_1) to R. Then R chooses a bit d at random and sends to S two pairs of ciphertext/plaintext $(C_0, M_0), (C_1, M_1)$. This is done such that only one pair is valid, i.e., C_d is an encryption of M_d under P_d, whereas C_{1-d} is a random ciphertext

for which R does not know the plaintext, and M_{1-d} is a random plaintext chosen independently.

Now S can decrypt C_c and test whether the result equals M_c. This with almost certainty determines d. Finally, S sends a bit $s = c \oplus d$ to R telling him whether $c = d$. If $c = d$, the parties will use this secret bit to communicate message bit m securely as $f = m \oplus c$. If $c \neq d$, we say that this attempt to communicate a bit has failed, and none of the bits c, d are used later.

The intuition now is that successful attempts can be faked by a simulator that chooses to know all secret keys and plaintexts involved, generates only valid ciphertext/plaintext pairs but leaves c undefined for the moment. Then, if for instance S is later corrupted, the simulator can choose to reveal the secret key corresponding to either P_0 or P_1 depending on the value it wants for c at that point. It will then be clear that the pair (C_c, M_c) is valid — the other pair is valid too, but the adversary cannot see this if the simulator can convincingly claim that P_{1-c} was chosen without learning the corresponding secret key.

A main part of the following is devoted to showing that if we define appropriately what it means to generate public keys and ciphertexts with no knowledge of the corresponding secrets, then this intuition is good.

4 Simulatable Public-Key Systems

Throughout the paper we will use the following notation. For a probabilistic algorithm $\mathcal{A}$ we will by $\mathcal{R}_\mathcal{A}$ denote a sufficiently large set $\{0,1\}^l$ from which the random bits for $\mathcal{A}$ are drawn. We let $r \leftarrow \mathcal{R}_\mathcal{A}$ denote a r drawn uniformly random from $\mathcal{R}_\mathcal{A}$, let $a \leftarrow \mathcal{A}(x,r)$ denote the result a of evaluating $\mathcal{A}$ on input x using random bits r, and denote by $a \leftarrow \mathcal{A}(x)$ a value a drawn from the random variable $\mathcal{A}(x)$ describing $\mathcal{A}(x,r)$ when r is uniform over $\mathcal{R}_\mathcal{A}$.

We now want to define a public-key encryption scheme where one can generate a public key without getting to know the matching secret key. So in addition to the normal key generation algorithm $\mathcal{K}$ that outputs a public and secret key (P, S), we assume that there is another algorithm which we call the oblivious public-key-generator $\tilde{\mathcal{K}}$ which outputs only a public key P with a distribution similar to public keys produced by $\mathcal{K}$. However, this condition is not sufficient to capture what we want. $\tilde{\mathcal{K}}$ could satisfy it by just running the same algorithm as $\mathcal{K}$ but output only P. We therefore also ask that based only on a public key P, there is an efficient algorithm $\tilde{\mathcal{K}}^{-1}$ that comes up with a set of random choices r' for $\tilde{\mathcal{K}}$ such that $P = \tilde{\mathcal{K}}(r')$ and that P, r' cannot be distinguished from a normal set of random choices and resulting output from $\tilde{\mathcal{K}}$. This ensures that whatever side information you get from producing P using $\tilde{\mathcal{K}}$, you could also compute efficiently from only P itself. In a similar way we can define what it means to produce a random ciphertext with no knowledge of the plaintext.

The property of being able to reconstruct the random bits used by an algorithm, we call invertible sampling. We define this notion first.

Definition 1 (Invertible sampling). *Let $A : X \times \{0,1\}^* \to Y$ be a PPT algorithm. We say that A has* invertible sampling *and that A is a* PPTIS *algorithm, if*

there exists a PPT random-bits-faking-algorithm $A^{-1} : Y \times X \to \{0,1\}^*$ *such that for all input* $x \in X$*, uniformly random bits* $r \leftarrow \mathcal{R}_A$*, output value* $y \leftarrow A(x,r)$*, and* fake random bits $r' \leftarrow A^{-1}(y,x)$ *the random variables* (x,y,r') *and* (x,y,r) *are computationally indistinguishable.*

Invertible sampling seems to be closely connected to non-committing encryption and adaptively secure computation in the non-erasure model.

As will be discussed further in chapter 6.2, the security of the non-committing encryption scheme in [9] relies on a invertible sampling property of the domains of the permutation. Also, the non-committing encryption scheme in [4], although not treated explicitly there, relies on the fact that you can invertible sample a quadratic residue in a specific group.

Invertible sampling is closely connected to adaptive security in models, where security is defined by requiring that an adversary's view of a real-life execution of a protocol can be simulated given just the data the adversary is entitled to, and where erasures are not allowed. Consider the protocol, where a party P_1 receives input x, computes $y \leftarrow f(x,r)$, where r is some uniformly random string, and outputs x. Assume that all other parties do nothing. After a corruption of P_1 a real-life adversary sees the input x, the output y, and the random bits r. By definition of security there exists an ideal-evaluation adversary $\mathcal{S}$, that given just x and y will output the same view, i.e. compute r' such that (x,y,r') are computationally indistinguishable from (x,y,r). Since $\mathcal{S}$ is a PPT algorithms it then follows that the protocol is secure iff f has invertible sampling. Why it is indeed meaningful to deem such a protocol insecure if f does not have invertible sampling, even though the protocol only has local computations, will not be discussed here.

Definition 2 (Simulatable public-key system). *Let* $(\mathcal{K},\mathcal{E},\mathcal{D},\mathcal{M})$ *be a public-key system with key-generator* $\mathcal{K}$*, encryption algorithm* $\mathcal{E}$*, decryption algorithm* $\mathcal{D}$*, message-generator* $\mathcal{M}$*, and security parameter* k *(*1^k *is implicitly given as input to all algorithms in the following.) We say that* $(\mathcal{K},\mathcal{E},\mathcal{D},\mathcal{M})$ *is a* simulatable public-key system *if besides fulfilling the usual requirements there exists PPTIS algorithms* $\tilde{\mathcal{K}}$ *and* $\mathcal{C}$*, called the* oblivious public-key-generator *resp. the* oblivious ciphertext-generator *such that the following holds.*

Oblivious public-key generation *For* $(P,S) \leftarrow \mathcal{K}$ *and* $\tilde{P} \leftarrow \tilde{\mathcal{K}}$ *the random variables* P *and* $\tilde{P}$ *are computationally indistinguishable.*

Oblivious ciphertext generation *For* $(P,S) \leftarrow \mathcal{K}$, $C_1 \leftarrow \mathcal{C}_P$ *and* $M \leftarrow \mathcal{M}_P$, $C_2 \leftarrow \mathcal{E}_P(M)$*, the random variables* (P,C_1) *and* (P,C_2) *are computationally indistinguishable.*

Semantic security *For* $(P,S) \leftarrow \mathcal{K}$*, and for* $i = 0,1$*:* $M_i \leftarrow \mathcal{M}_P$*,* $C_i \leftarrow \mathcal{E}_P(M_i)$*, the random variables* (P,M_0,M_1,C_0) *and* (P,M_0,M_1,C_1) *are computationally indistinguishable.*

5 Non-Committing Encryption

Non-committing encryption was defined in [9] as the problem of communicating a bitstring from a party S to party R in a n-party network with insecure authenticated channels between all parties. It is required that the protocol for doing this is secure against an adaptive adversary, who can corrupt up to $n-1$ parties.

We introduce and provide examples of protocols adhering to a stronger notion of non-committing encryption which is resilient against a corruption of all parties and which involves only the communicating parties.

Definition 3 (Strong non-committing encryption). *Let $f(m, \epsilon) = (\epsilon, m)$ be the two-party function for communicating a message $m \in \{0,1\}^*$. Let π be a two-party protocol. We say that π is a strong non-committing encryption scheme if it 2-adaptively securely computes f.*

A reason for preferring a protocol meeting definition 3 is first of all that a protocol meeting the strong notion allows two parties to communicate independently of the other parties. We can think of such a protocol as a channel between two parties maintained by just these parties. This provides more flexibility for use in sparse network topologies and with arbitrary adversary structures.

Many proposals for the definition of secure multiparty computation has appeared in the literature presently culminating in the proposal of [8] which as the first definition allows for general security preserving modular composition of protocols in the computational setting. We will use this model of secure multiparty computation.

In general the model defines an ideal-evaluation of a function f and requires that whatever a PPT real-life adversary $\mathcal{A}$ might obtain from attacking a real-life execution of the protocol π a corresponding ideal-evaluation adversary $\mathcal{S}$ could obtain from attacking only the ideal-evaluation.

In our case the ideal-evaluation is functionally as follows. There are three active parties, all PPT algorithms. The sender S, the receiver R, and the adversary $\mathcal{S}$. The sender and receiver shares a *secure* channel and S simply sends the message m to R. The adversary sees no communication, but can corrupt the parties adaptively. If so he learns m (either as the senders input or the receivers output) and can control the corrupted party for the remaining evaluation. I.e. if S is corrupted before sending m the adversary might send a different message.

In the real-life execution the adversary $\mathcal{A}$ sees all communication, and if he corrupts a party he receives that parties input and output (here m) *and* that parties random bits. All communication, inputs values, and all random bits are enough that the adversary can reconstruct the entire execution history of the corrupted party. This is what captures the non-erasure property of the model.

We then define security by requiring that for any real-life adversary $\mathcal{A}$ there exists an ideal-evaluation adversary $\mathcal{S}$, such that the collective output of all uncorrupted parties and $\mathcal{S}$ after attacking an ideal-evaluation of sending m is distributed computationally indistinguishable from the collective output of all

uncorrupted parties and $\mathcal{A}$ after attacking a real-life execution of the protocol with input m.

A complete definition and a summary of previous definitional work appears in [8]. A sketch of the part of the model used in this paper appears in our technical report [11].

5.1 The Main Idea

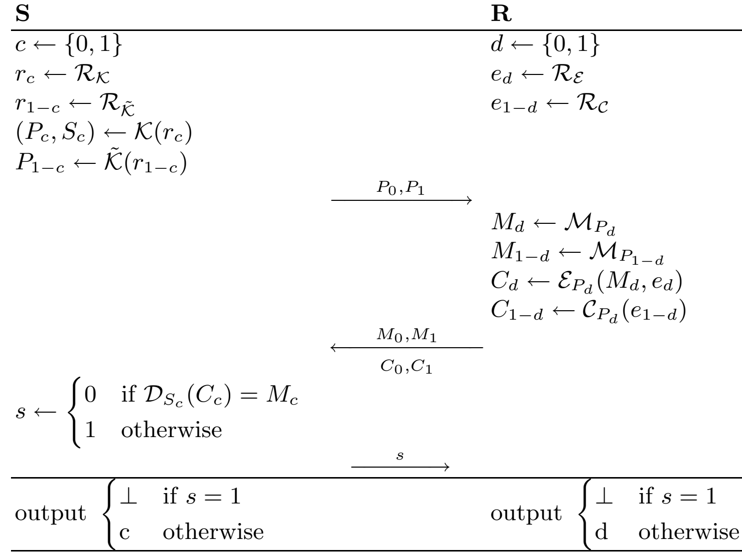

Fig. 1. One attempt to establish a shared random bit.

The main idea in the protocol is — like in all previous proposals — that we have our parties learn less information than is actually possible. This opens the possibility that a simulator can choose to learn full information and exploit this to its advantage. The main building block of the protocol, which we call an attempt, is sketched in Fig. 1.

Let r_S and r_R be the random inputs of S resp. R. We write the values obtained by an attempt as

$$\text{Attempt}(r_S, r_R) = (r_c, P_c, M_c, e_c, C_c), S_c, (r_{1-c}, P_{1-c}, M_{1-c}, e_{1-c}, C_{1-c}), (c, d, s)$$

Let Attempt denote the random variable describing $\text{Attempt}(r_S, r_R)$ when r_S and r_R are chosen uniformly random. Let Attempt_i for $i = 0, 1$ denote the distribution of Attempt under the condition that $s = i$. An attempt where $s = 0$ is called a successful attempt and an attempt where $s = 1$ is called a failed attempt.

For later use in the simulator and for illustration of the main idea we now show how we can produce a distribution computationally indistinguishable from that of Attempt_0, but where the common value $b = c = d$ of the shared secret bit can later be changed. We say that the simulation is non-committing to b.

Let SimSuccess be the values produced as follows: $s \leftarrow 0, i = 0, 1 : \ r_i \leftarrow \mathcal{R}_{\mathcal{K}}$, $(P_i, S_i) \leftarrow \mathcal{K}(r_i)$, $M_i \leftarrow \mathcal{M}$, $e_i \leftarrow \mathcal{R}_{\mathcal{E}}$, $C_i \leftarrow \mathcal{E}_{P_i}(M_i, e_i)$. The only difference compared to Attempt_0 is that in SimSuccess, we choose to learn the corresponding private key of P_{1-c}, choose C_{1-d} as an encryption of M_{1-d}, and do not fix c and d yet.

For patching successful attempts we define the function $\text{Patch}(A, b)$, which for an element A drawn from SimSuccess and a bit $b \in \{0, 1\}$ produces values similar to those in Attempt_0 by computing c and d as $c \leftarrow b$, $d \leftarrow b$, and patching r_{1-c} and e_{1-d} by $r'_{1-c} \leftarrow \tilde{\mathcal{K}}^{-1}(P_{1-c})$, $e'_{1-d} \leftarrow \mathcal{C}^{-1}(C_{1-d}, P_{1-d})$.

Let $\text{Patch} = (r_c, P_c, M_c, e_c, C_c), S_c, (r'_{1-c}, P_{1-c}, M_{1-c}, e'_{1-c}, C_{1-c}), (c, d, s)$ denote the random variable describing $\text{Patch}(A, b)$ when A is drawn randomly from SimSuccess and b is chosen uniformly random from $\{0, 1\}$.

Lemma 1. *The distribution of* Patch *is computationally indistinguishable from the distribution of* Attempt_0.

Proof: Let b denote the common value of c and d and observe that $\Pr[b = 0]$ and $\Pr[b = 1]$ is negligible close to $\frac{1}{2}$ in both Patch and Attempt_0. It is therefore enough to show that the conditional distributions under $b = 0$ and $b = 1$ are computationally indistinguishable.

For fixed b the variables c, d, and s are constants and has the same values in the two distributions, so we can exclude them from the analysis. Further more $(r_c, P_c, M_c, e_c, C_c), S_c$ can be seen to have the same distribution in the two distributions and is independent of $(r_{1-c}, P_{1-c}, M_{1-c}, e_{1-c}, C_{1-c})$, so all that remains is to show that these $(1 - c)$-values are distributed computationally indistinguishable in Attempt_0 and Patch. In Attempt_0 these values are distributed as

$$(\tilde{r} \leftarrow \mathcal{R}_{\tilde{\mathcal{K}}}, \tilde{P} \leftarrow \tilde{\mathcal{K}}(\tilde{r}), M \leftarrow \mathcal{M}_{\tilde{P}}, e \leftarrow \mathcal{R}_{\mathcal{C}}, C \leftarrow \mathcal{C}_{\tilde{P}}(e)) \tag{1}$$

and in Patch they are distributed as

$$(r', P, M \leftarrow \mathcal{M}_P, e', C \leftarrow \mathcal{E}_P(M, e)) \tag{2}$$

where $r \leftarrow \mathcal{R}_{\mathcal{K}}$, $(P, S) \leftarrow \mathcal{K}(r)$, $r' \leftarrow \tilde{\mathcal{K}}^{-1}(P)$ and $e \leftarrow \mathcal{R}_{\mathcal{E}}$, $e' \leftarrow \mathcal{C}^{-1}(C, P)$. That these distributions are computationally indistinguishable follows by a hybrids argument, going from (1) to (2) using (in this order) the oblivious public-key generation including the invertible sampling of $\tilde{\mathcal{K}}$, the oblivious ciphertext generation including the invertible sampling of $\mathcal{C}$, and finally the semantic security. For more details see the technical report [11]. ■

Why Failed Attempts Cannot be Simulated without Committing Consider the situation where $c \neq d$. The secret key S_c is always known by S. If this

key becomes known to the adversary by corrupting S, he can check whether $\mathcal{D}_{S_c}(C_{1-d}) \neq M_{1-d}$, as it should be with high probability. The simulator can therefore pick at most one message/encryption pair such that C is an encryption of M. On the other hand the adversary when corrupting R expects to see a value d and random bits e_d such that $C_d \leftarrow \mathcal{E}_{P_d}(M_d, e_d)$. Thus at least one message/encryption pair should be correct. All in all exactly one pair is correct, which commits the simulator to d (and thus c).

5.2 The Full Protocol

We will here analyse the protocol in Fig. 2, where we execute attempts in sequence until we have a successful one and then use the shared secret value b of c and d to communicate a message bit m as $f = m \oplus b$.

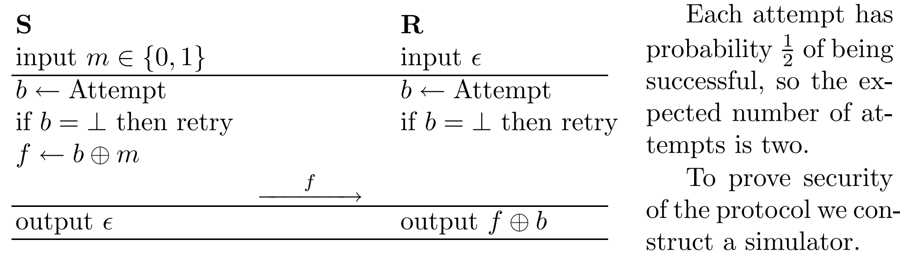

Fig. 2. Sequential 1-bit protocol.

Each attempt has probability $\frac{1}{2}$ of being successful, so the expected number of attempts is two.

To prove security of the protocol we construct a simulator.

5.3 The Simulator

Let $\mathcal{A}$ be any real-life adversary. We construct a corresponding ideal-evaluation adversary $I(\mathcal{A})$ as follows. The ideal-evaluation adversary $I(\mathcal{A})$ initialises $\mathcal{A}$ with a sufficiently large random string $r_{\mathcal{A}}$. The real-life adversary $\mathcal{A}$ will now start attacking. It expects to attack a real-life execution, but $I(\mathcal{A})$ is attacking an ideal-evaluation. We describe how to handle this.

The Basic Simulation As long as $\mathcal{A}$ does not corrupt any parties $I(\mathcal{A})$ proceeds as follows. Before simulating each attempt decide whether the attempt should be a success or should fail by drawing s uniformly random from $\{0,1\}$.

If $s = 1$ then start by preprocessing values to be revealed to $\mathcal{A}$. Simply execute the protocol for a failed attempt. I.e. draw c uniformly random from $\{0,1\}$, set $d = 1 - c$, and then execute the attempt protocol in Fig. 1. This provides the values $(r_c, P_c, M_c, e_c, C_c)$, S_c, $(r_{c-1}, P_{c-1}, M_{c-1}, e_{c-1}, C_{c-1})$, $(c, d = 1-c, s = 1)$. The adversary expects to see all communication in a real-life execution, so show him (P_0, P_1), (M_0, M_1, C_0, C_1), and s, in that order — we will later deal with the issue of how $I(\mathcal{A})$ should act on corruption requests from $\mathcal{A}$.

If $s = 0$, then $I(\mathcal{A})$ simulates a successful attempt. Again values for the communication is preprocessed. This time by running the algorithm $A \leftarrow$ SimSuccess. This provides values for all communication except f. We pick f uniformly random from $\{0,1\}$ and reveal the communication to $\mathcal{A}$ as above.

When a successful attempt has been simulated the actual simulation is over, but $I(\mathcal{A})$ keeps interacting with $\mathcal{A}$, which might still corrupt more parties — we do return to this right ahead. At some point $\mathcal{A}$ terminates with some output value, which we take to be the output value of $I(\mathcal{A})$.

Dealing with Corruption Requests Below we describe how to handle the first corruption request. We look at two points, where the first corruption might take place. During the failed attempts and during the successful attempt. We prove that in either case $I(\mathcal{A})$ can patch the internal simulated state of S and R and simulate the corruption such that the entire state of S, R, and $\mathcal{A}$ is computationally indistinguishable from the state that would have been produced by running a real-life execution on the same input with adversary $\mathcal{A}$.

From this it follows that the simulator can then complete the simulation by just running the remaining honest party according to the real-life protocol with the simulated state as a starting point. Since the starting point is computationally indistinguishable from that of a real-life execution at the same point of execution and all participating algorithms are PPT it follows directly from the definition of computational indistinguishability that the final output of $\mathcal{A}$, and thereby $I(\mathcal{A})$, will be computationally indistinguishable from the output of $\mathcal{A}$ produce by an execution of the real-life protocol [2].

If $\mathcal{A}$ corrupts a party during the simulation of a failed attempt $I(\mathcal{A})$ corrupts the corresponding party in the ideal-evaluation and learns m. Observe that the simulated communication values *and* all preprocessed internal values in failed attempts are distributed *identically* to the values produced by a real-life execution. Therefore after obtaining m the simulator $I(\mathcal{A})$ can just pass this along to $\mathcal{A}$ and the obtained global state is distributed *identically* to that of a real-life execution.

If $\mathcal{A}$ corrupts a party during the simulation of a successful attempt, we again have a number of cases as there is three rounds of communication. We first look at the case where the corruption occurs after s and f have been communicated. Here $I(\mathcal{A})$ again corrupts the same party in the ideal-evaluation, obtains m, and passes it on to $\mathcal{A}$. Now $I(\mathcal{A})$ must decide on values for c and d. We pick the common value $b \leftarrow m \oplus f$. This value is consistent with all other values since with $c = b$ and $d = b$ we have that $f = m \oplus c$ and $m = f \oplus d$ as required. The simulator now patches the preprocessed values using $\text{Patch}(A, m \oplus f)$ and hands out the patched values thus produced to $\mathcal{A}$. Observe that $m \oplus f$ is uniformly distributed over $\{0, 1\}$ as we picked f uniformly random. It then follows directly from lemma 1 and the fact that c and d is chosen consistent with f and m that the global state of S, R, and $\mathcal{A}$ is computationally indistinguishable from that

[2] Note that what allows this simple argument is that in contrast to simulators for more involved protocols not only have we obtained that the values revealed to $\mathcal{A}$ up to the first corruption is computationally indistinguishable from those of a real-life execution. We have managed to produce a complete global state of communication and the state of corrupted *and* uncorrupted parties that is computationally indistinguishable from that of a real-life execution.

which would have been produced from an execution of the real-life protocol on the same inputs.

If the corruption occurs before f is revealed to $\mathcal{A}$, just obtain m as before and patch the preprocessed values with a uniformly random common value b for c and d. The value of f will then be given by the real-life protocol when $I(\mathcal{A})$ starts the execution of the remaining honest party. Earlier corruptions are handled similarly.

Theorem 1. *If simulatable public-key systems exist, then the protocol in Fig. 2 is a strong non-committing encryption scheme for communication one bit.*

Proof: We have to prove that for all real-life adversaries $\mathcal{A}$ there exists an ideal-evaluation adversary $\mathcal{S}$ such that the output of $\mathcal{A}$ after attacking the real-life protocol is computationally indistinguishable from the output of $\mathcal{S}$ after attacking an ideal-evaluation of the same inputs.

Given $\mathcal{A}$ we simply set $\mathcal{S} = I(\mathcal{A})$ an the claim follows from the above analysis of $I(\mathcal{A})$. ■

Theorem 2. *If simulatable public-key systems exist, then strong non-committing encryption schemes exist. The scheme requires 3 messages to communicate k encrypted bits, where k is the security parameter. The total amount of communication is $O(k)$ public keys, $O(k)$ encryptions of a k-bit plaintext (in the original scheme), and k bits.*

Proof: It follows directly from the Markov inequality that $a = 4k$ parallel attempts will give k successful ones for communication except with probability $\exp(-\frac{k}{2})$, which is certainly negligible in k. This protocol uses three rounds for the attempts and we can communicate f in round three together with s. We thereby obtain the claimed round complexity. The claimed communication complexity is trivially correct.

Since the simulator for attempts does not rewind the real-life adversary $\mathcal{A}$, we can obtain a simulator for the parallel protocol by simply running a 'copies' of the simulator for the attempts in parallel. See the technical report [11] for more details. ■

6 Implementations

The following theorem provides the first example of a simulatable public-key system.

Theorem 3. *The ElGamal public-key system is simulatable assuming that it it semanticly secure.*

Proof: Recall that a public key is a triple (p, g, h), where p is a prime such that the discrete log problem in $\boldsymbol{Z}_p^*$ is intractable, $\langle g \rangle = \boldsymbol{Z}_p^*$, and $h = g^x$ for some random x, which is the private key. Now simply let the oblivious public-key-generator pick h directly in $\boldsymbol{Z}_p^*$ without learning its discrete log base g and let

$\tilde{\mathcal{K}}^{-1}(p,g,h) = (r_p, r_g, r_h)$, where r_p, r_g, and r_h are random bits similar to those used to pick p, g, resp. h.

How to reconstruct r_p depends of course on the algorithm used to pick p. For simplicity, say that we pick p by drawing random numbers in some interval I until we get a number that tests to primality by some probabilistic test. We will then have to reconstruct, from p, a distribution similar to the prefix of numbers that were not primes. This can trivially be done by drawing random numbers in I until a prime is found and use the prefix of non-primes and the random bits used to test them non-prime. The value r_p is set to be these bits, p, and bits used to test p prime using the primality test. This value r_p is trivially distributed computationally indistinguishable from the bits originally used to pick p. The oblivious public-key generation is then trivially fulfilled. The values r_g and r_h are trivial to reconstruct if g and h is chosen in a natural way.

A message $x \in \boldsymbol{Z}_p^*$ is encrypted as (g^k, xh^k), where k is chosen uniformly random in $\boldsymbol{Z}_{p-1}$. It is obvious, that a ciphertext can be generated obliviously as (y_1, y_2), where y_1 and y_2 are picked uniformly random and independent in $\boldsymbol{Z}_p^*$. Invertible sampling is trivial. ∎

6.1 Trapdoor Permutations

Before presenting the next example of a simulatable public-key system, we define the concept of a simulatable collection of trapdoor permutations and prove that the existence of such a collection implies the existence of simulatable public-key systems.

We first recall the standard definition of collections of trapdoor permutations:

Definition 4 (Collection of trapdoor permutations). *We call $(I, F, \mathcal{G}, \mathcal{X})$ a collection of trapdoor permutations with security parameter k, if I is an infinite index set, $F = \{f_i : D_i \to D_i\}_{i \in I}$ is a set of permutations, the index/trapdoor-generator $\mathcal{G}$ and the domain-generator $\mathcal{X}$ are PPT algorithms, and the following hold:*

Easy to generate and compute *$\mathcal{G}$ generates pairs of indices and trapdoors, $(i, t_i) \leftarrow \mathcal{G}(1^k)$, where $i \in I \cap \{0,1\}^{p(k)}$ for some fixed polynomial $p(k)$. Furthermore, there is a polynomial time algorithm which on input $i, x \in D_i$ computes $f_i(x)$.*

Easy to sample domain *$\mathcal{X}$ samples elements in the domains of the permutations, i.e. $x \leftarrow \mathcal{X}(i)$, where x is uniformly random in D_i.*

Hard to invert *For $(i, t_i) \leftarrow \mathcal{G}(1^k)$, $x \leftarrow \mathcal{X}(i)$, and for any PPT algorithm A the probability that $A(i, f_i(x)) = x$ is negligible in k.*

But easy with trapdoor *There is a polynomial time algorithm which on input $i, t_i, f_i(x)$ computes x, for all $x \in D_i$.*

The next definition, of simulatable collections, is built along the lines of the definition of simulatable public-key systems. It basically defines a collection of trapdoor permutations where in addition it is easy to generate a permutation f in

the collection without getting to know the trapdoor. Further more we need that the domain of the trapdoors has invertible sampling. This is to allow oblivious ciphertext generation.

Invertible sampling is trivial if the domain of f is, for instance, the set of k-bit strings and sampling is done in the natural way. But it may in general be an extra requirement which, however, seems to be necessary for any application of the kind we consider here. It is easy to construct artificial domains without invertible sampling, but all collections of trapdoor permutations we know of have domains with invertible sampling.

Definition 5 (Simulatable collection of trapdoor permutations). *Let $(I, F, \mathcal{G}, \mathcal{X})$ be a collection of trapdoor permutations with security parameter k. We say that $(I, F, \mathcal{G}, \mathcal{X})$ is a* simulatable collection of trapdoor permutations *with* oblivious index-generator $\tilde{\mathcal{G}}$*, if $\tilde{\mathcal{G}}$ and $\mathcal{X}$ are PPTIS algorithm and the random variables i and $\tilde{i}$ are computationally indistinguishable, where $(i, t_i) \leftarrow \mathcal{G}$ and $\tilde{i} \leftarrow \tilde{\mathcal{G}}$.*

Given F a collection of trapdoor permutations one can construct a semantically secure public-key system using the construction in [7]. We review the construction here and observe that it preserves simulatability.

Let B be a hard-core predicate of the collection of trapdoor permutations. If no such B is known one can construct a new simulatable collection of trapdoor permutations following the construction in [7]. The key-generator is set to be $\mathcal{K} = \mathcal{G}$, i.e. for $(i, t_i) \leftarrow \mathcal{G}$ we set $(P, S) = (i, t_i)$. The message space can be set to $\mathcal{M} = \{0,1\}^{p(k)}$ for any polynomial $p(k)$ and the ciphertext space for $P = i$ is $\mathcal{M} \times D_i$, where D_i is the domain of f_i.

Let $X(i, x, n) = B(x)B(f_i(x))B(f_i^2(x)) \dots B(f_i^{n-1}(x))$ be the usual pseudo-random string generated from x. Then the encryption of $m \in \mathcal{M}$ under i is $(m \oplus X(i, x, |m|), f_i^{|m|}(x))$ for random $x \leftarrow \mathcal{X}(i)$. The decryption is trivial given t_i. To pick such a ciphertext obliviously for a given key P generate $m \leftarrow \mathcal{M}$ and $x \leftarrow \mathcal{X}(i)$ and let $C = (m, x)$. This will be distributed exactly as $C \leftarrow \mathcal{E}_P(m')$ for $m' \leftarrow \mathcal{M}$. Invertible sampling is given by $\mathcal{C}^{-1}(i, m, x) = (m, \mathcal{X}^{-1}(i, x))$.

The oblivious public-key generation is given by setting $\tilde{\mathcal{K}} = \tilde{\mathcal{G}}$ and $\tilde{\mathcal{K}}^{-1} = \tilde{\mathcal{G}}^{-1}$ and the semantic security is proven in [7].

Theorem 4. *Let $\mathcal{F} = (I, F, \mathcal{G}, \mathcal{X})$ be a simulatable collection of trapdoor permutations and let $E_{\mathcal{F}} = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{M})$ be the public-key system described above. Then $E_{\mathcal{F}}$ is simulatable.*

We proceed to construct a simulatable collection of trapdoor permutations based on RSA. We cannot use the standard collection of RSA-trapdoors as it has not been proven to have oblivious public-key generation. If the oblivious public-key generation learns the factorisation of n, the random-bits-faking-algorithm would have to factor n, which is hopefully hard. If the oblivious public-key generation does not learn the factorisation of n it would have to test in PPT whether n is a factor of two large primes, which we do not know how to do. We therefore need a modification.

Assumption 1 *Let* $I = \{(n,e)|n = pqr,\ p,\ q$ *are primes and* $|p|, |q| \geq k,\ |n| = k \log k$*, and* $n < e < 2n$ *is a prime*$\}$*. Here,* k *is as usual the security parameter. For* $(n,e) \in I$ *let* $t_{(n,e)} = d$ *where* $ed = 1 \mod \phi(n)$*. Let* $f_{(n,e)} : \boldsymbol{Z}_n^* \to \boldsymbol{Z}_n^*, x \mapsto x^e \bmod n$*. Then* $F = \{f_i\}_{i \in I}$ *is a collection of trapdoor permutations.*

Observe, that there is a non-negligible chance that a random integer n contains two large primefactors. I.e. if we pick n at random and e as a prime larger than n, then $x \mapsto x^e \bmod n$ is a weak trapdoor permutation over $\boldsymbol{Z}_n^*$ (relative to assumption 1.) The same observation was used in [9], where they refer to general amplification results[21, 13] to obtain a collection of strong trapdoor permutations from this collection of weak ones. Here we apply an explicit amplification procedure, which is slightly more efficient, and prove that it gives us a simulatable collection of trapdoor permutations.

Let l be an amplification parameter, which we fix later.

An index with corresponding trapdoor is given by $i = (e, n_1, \dots, n_l)$ and $t_i = (d_1, \dots, d_l)$, where e is a $(k \log k)$-bit random prime and for $j = 1, \dots, l$ the number n_j is a uniformly random $(k \log k)$-bit number and $d_j = e^{-1} \mod \phi(n_j)$. To compute d_j the key-generator $\mathcal{G}$ must generate uniformly (or indistinguishably close to uniformly) random n_j in such a way that $\phi(n_j)$ is known. In [2] it was shown how to do this.

An oblivious index $(e, n_1, \dots, n_l) \leftarrow \tilde{\mathcal{G}}$ is simply generated by picking e as before and picking the n_j uniformly random. The only problem for $\tilde{\mathcal{G}}^{-1}$ in faking bits for the index $(e, n_1, \dots, n_l)$ is the prime e. On how to do this see the proof of theorem 3.

The domain for the index $i = (e, n_1, \dots, n_l)$ will be $D_i = \prod_{j=1}^{l} \boldsymbol{Z}_{n_j}^*$ and the corresponding permutation will be $f_{(e,n_1,\dots,n_l)}(x_1, \dots, x_l) = (x_1^e \bmod n_1, \dots, x_l^e \bmod n_l)$. Since e is relatively prime to all n_j our functions are indeed permutations and are invertible in PPT using the trapdoor information $t_i = (d_1, \dots, d_l)$.

We pick a uniformly random element x from D_i by picking a uniformly random element x_j from each $\boldsymbol{Z}_{n_j}^*$. These elements should be chosen in a way that allows $\mathcal{X}^{-1}$ to reconstruct the random bits used. One way is to pick uniformly random elements from $\boldsymbol{Z}_{n_j}$ until an element from $\boldsymbol{Z}_{n_j}^*$ is found. This gives us $\mathcal{X}^{-1}$ by following the construction for primes — see the proof of theorem 3.

What remains is to prove the one-wayness of our collection. In [18] the probability that the i'th largest primefactor of a random number n is larger than n^c for a given constant c is investigated. It is shown to approach a constant as n approaches infinity. In particular, the probability that the second largest primefactor is smaller than n^c is approximately linear for small c, in fact it is about $2c$ for $c \leq 0.4$. It follows that the probability that a number of length $k \log k$ bits has its second largest prime factor shorter than k bits is $O(1/\log k)$. If we set l to $\log k$, we then have that the probability that there does not exist $j \in \{1, \dots, l\}$ such that $(n_j, e) \in I$, where I is the index set of assumption 1, is $O((\frac{1}{\log k})^{\log k})$ and so is negligible. So we have:

Theorem 5. *Under assumption 1, the set* $SRSA = \{f_i : D_i \to D_i\}$ *is a simulatable collection of trapdoor permutations.*

We note that $(\frac{1}{\log k})^{\log k}$ is only slightly below what is needed to be negligible. To obtain a security preserving[13] amplification we could use k k-bit moduli. Another approach would be to remove the need for amplification by finding an invertible way to produce integers with two large primefactors and use just one such modulus for encryption.

6.2 Doing without Oblivious Index Generation

We now proceed to prove that one can do without the oblivious index generation.

We basicly remove the oblivious index generation assumption by using the key generation protocol from [9] applying a fix and a twist. The fix is necessary as we have found an attack against the protocol used in [9]. The twist is applied to remove the common-domain assumption which is needed by the construction in [9].

The Key Generation Protocol In [9] a non-committing encryption scheme was built consisting of two phases. The first phase is a key generation protocol which is intended to create a situation, where players S and R share two trapdoor permutations from what is called a common-domain trapdoor system. Moreover, S knows exactly one of the corresponding trapdoors, and if S remaines honest in this phase, a simulator is able to make a simulated computation, where both trapdoors are learned and which can later (in case S is corrupted) be convincingly patched to look as if either of the trapdoors were known to S. One immediate consequence is that the adversary must not know which of the two trapdoors is known to S, before corrupting S.

The key generation requires participation of all n parties of the protocol and proceeds as follows: Each player P_i chooses at random two permutations (g_0^i, g_1^i) and send these to S. Next S chooses $c = 0$ or 1 at random, and execute the oblivious transfer (OT) protocol of [14] with P_i as sender using the trapdoors of (g_0^i, g_1^i) as input and S as receiver using c as input, and such that S receives the trapdoor of g_c^i. The OT protocol of [14] has a non-binding property that allows a simulator to learn both trapdoors when it is playing S's part and later to claim that either trapdoor was received.

In the above, there is no guarantee that P_i really uses the trapdoors of (g_0^i, g_1^i) as input to the OT, but, as pointed out in [9] one may assume that the trapdoor of a permutation consists of all inputs required to generate it so that S can verify what he receives. Finally, S publishes the subset A of players from whom he got correct trapdoors, and we define f_0 to be the composition of the permutations $\{g_0^i\}_{i \in A}$ in some canonical order, and similarly for f_1.

The Attack and a Fix We describe an attack against the above key generation protocol. If S is still honest, but P_i is corrupt, the adversary may choose to let P_i use as inputs to the OT a correct trapdoor for g_a^i but garbage for g_b^i. When the adversary sees the set A he can then determine the value of c. If $i \notin A$ the sender must have chosen $c = b$ and detected P_is fraud. If $i \in A$ then the sender must

have chosen $c = a$. In any case the adversary learns c and in $\frac{1}{2}$ of the cases even without being detected. But this is a piece of information that the adversary should not be able to get. The simulator's freedom to set c after corruption is exactly what makes the simulation go through.

We solve this by requiring that a sender in an OT always proves in zero-knowledge to the receiver he inputs correct information to the OT. I.e. prove that there exists r such that $(g, t_g) = \mathcal{G}(r)$ and that there exists a bitstring r' such that (g, t_g, r') is the inputs to the OT (r' being the random bits used in the OT.) Since this is an NP statement and R knows both witnesses r and r', this is always possible to prove[14]. This will imply that except with negligible probability, P_i will have to supply correct trapdoors to both permutations or be disqualified. Normally, the use of such ZK proofs leads to problems against adaptive adversaries because of the rewinding needed to simulate the proofs. However, in this protocol, it happens to be the case that the simulator never needs to "prove" a statement for which it doesn't know a witness, and so rewinding is not needed.

The Twist Having executed all the OT's, S would in the protocol from [9] compose the permutations from honest parties to obtained two permutations, one with a known trapdoor and one with an unknown trapdoor. This requires and produces common-domain trapdoors. Instead of composing we simply concatenate.

Let $g_c^1, \ldots, g_c^l$ be the permutations that was correctly received and for which the corresponding trapdoor was received. From these permutations S defines a new permutation f_c, where $f_c(x^1, \ldots, x^l) = (g_c^1(x^1), \ldots, g_c^l(x^l))$. Let B be a hard-core predicate for the collection of trapdoor permutations used. Then $B(x^1, \ldots, x^l) = \bigoplus_{i=1}^{l} B(x^i)$ is a hard-core predicate for f_c. Let $x^1, \ldots, x^l$ be random and let $X(g^i, x^i, n) = B(x^i)B(g^i(x^i))B((g^i)^2(x^i)) \ldots B((g^i)^{n-1}(x^i))$ be the usual pseudo-random string generated from x^i. Then the encryption of $m \in \{0,1\}^*$ under f_c using the above hard-core predicate is seen to be $((g^1)^{|m|}(x^1), \ldots, (g^l)^{|m|}(x^l)), m \oplus X(g^1, x^1, |m|) \oplus \ldots \oplus X(g^l, x^l, |m|)$. Similar for f_{1-c}.

In the following we call the $(g_1, \ldots, g_l)$ tuples public keys and the $(t_{g_1}, \ldots, t_{g_l})$ tuples private keys to distinguish from the individual permutations and trapdoors.

Using the Key Generation in Our Protocol After an execution of the key generation protocol an honest S has two public keys where he knows the private key to exactly one of them, but where the adversary cannot tell which one he knows. This is exactly the scenario that the first round of our protocol described earlier creates. Thus, one attempt to exchange a secret key can be done by running the key generation protocol followed by our communication phase. Our technical report [11] contains an analysis of the security. We sketch it here.

As before all failed attempts are simulated by simply following the real-life protocol. In the simulation of the successful attempt the simulator makes sure that it knows both private keys by learning all involved trapdoors: for each OT, if the sender is honest it chooses the trapdoors itself, if not, it chooses to learn

both trapdoors during the OT (this succeeds except with negligible probability by the ZK proofs we introduced.) On corruption the simulator can patch the view of S to be consistent with either private key being learned.

To show this simulation is indistinguishable from a real execution we observe that the only event in which there is a difference between the distributions is when R or S are corrupted after the message is sent. Here, the adversary will see a message/ciphertext pair which is valid w.r.t. to a given public key in the simulation but is invalid in a real execution. Since the adversary cannot corrupt all players, there is at least one involved trapdoor he does not know, so he should not be able to tell the difference. To prove this, we can take a permutation f with unknown trapdoor, and choose a random player P_i. We then run the simulation pretending that P_i chose f as one of its inputs to the OT and hoping that the adversary will not corrupt P_i but will corrupt S or R later. If simulation and execution can be distinguished at all, this must happen with non-negligible probability. It now follows that a successful distinguisher must be able to break encryption using f as public key.

Theorem 6. *If there exist collections of trapdoor permutations for which the domains have invertible sampling, then non-committing encryption schemes exist.*

We note that the OT protocol which we use as an essential tool is itself based on trapdoor permutations. Moreover, in order for the OT to be non-committing, the domain of permutations must have invertible sampling. This property is also necessary in the original key generation protocol from [9], where also a common-domain property was needed, so assuming only invertible sampling is a weaker assumption. Further more, the discussion in chapter 4 of invertible sampling might indicate, that a protocol using ideas similar to those presented in this paper will need the invertible sampling property of crucial domains sampled *or* use a n-party protocol for sampling the domains, as we did for the key space in this chapter.

References

1. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, 2–4 May 1988.
2. Eric Bach. How to generate factored random numbers. *SIAM Journal on Computing*, 17(2):179–193, April 1988.
3. D. Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 377–391, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.
4. D. Beaver. Plug and play encryption. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 75–89, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.
5. D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In Rainer A. Rueppel, editor, *Advances in Cryptology - EuroCrypt '92*, pages 307–323, Berlin, 1992. Springer-Verlag. Lecture Notes in Computer Science Volume 658.

6. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In ACM [1], pages 1–10.
7. M. Blum and S. Goldwasser. An efficient probabilistic public key encryption scheme which hides all partial information. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology: Proceedings of Crypto '84*, pages 289–302, Berlin, 1985. Springer-Verlag. Lecture Notes in Computer Science Volume 196.
8. Ran Canetti. Security and composition of multi-party cryptographic protocols. Obtainable from the Theory of Cryptography Library, august 1999.
9. Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 639–648, Philadelphia, Pennsylvania, 22–24 May 1996.
10. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In ACM [1], pages 11–19.
11. Ivan B. Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. Research Series RS-00-6, BRICS, Department of Computer Science, University of Aarhus, March 2000.
12. Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *33rd Annual Symposium on Foundations of Computer Science*, pages 427–436, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.
13. Oded Goldreich, Russell Impagliazzo, Leonid Levin, Ramarathnam Venkatesan, and David Zuckerman. Security preserving amplification of hardness. In *31st Annual Symposium on Foundations of Computer Science*, volume I, pages 318–326, St. Louis, Missouri, 22–24 October 1990. IEEE.
14. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 174–187, Toronto, Ontario, Canada, 27–29 October 1986. IEEE.
15. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, 25–27 May 1987.
16. IEEE. *23rd Annual Symposium on Foundations of Computer Science*, Chicago, Illinois, 3–5 November 1982.
17. Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: introducing concurrency, removing erasures. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 221–242, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.
18. D. E. Knuth and L. Trabb Pardo. Analysis of a simple factorization algorithm. *Theoretical Computer Science*, 3(3):321–348, 1976.
19. S. Micali and P. Rogaway. Secure computation. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 392–404, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.
20. Andrew C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science* [16], pages 160–164.
21. Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science* [16], pages 80–91.